**AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY**

**AMITY UNIVERSITY UTTAR PRADESH**



**Artificial Intelligence Practical File**

**Submitted by:** Madhur Gusain A2305218572 7CSE – 9X

**Submitted to:**

Dr. Aakankshi Gupta

# INDEX

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.**  **No** | **Category of Assignment** | **Code** | **Prog. No.** | **Name of Program** | **Date of Allotment** | **Submission Date** | **Max Mark** | **Marks Obtained** | **Signature of Faculty** |
| **1.** | **Mandatory Experiment** | **LR (10)** | **1** | Basics of Python programming |  | 21-07-2021 | **1** |  |  |
| **2.** | **Mandatory Experiment** | **2** | Water-jug Problem by BFS |  | 28-07-2021 | **1** |  |  |
| **3.** | **Mandatory Experiment** | **3** | 1. Graph colouring Problem 2. Depth-first Search |  | 04-08-2021 | **1** |  |  |
| **4.** | **Mandatory Experiment** | **4** | 1. Best-first Search 2. A\* algorithm |  | 11-08-2021 | **1** |  |  |
| **5.** | **Mandatory Experiment** | **5** | Constraint satisfaction algorithm |  | 18-08-2021 | **1** |  |  |
| **6.** | **Mandatory Experiment** | **6** | Tic-Tac-Toe by Mini-max algorithm |  | 31-08-2021 | **1** |  |  |
| **7.** | **Open-ended Experiment** | **7** | 4-Queens Problem |  | 07-09-2021 | **1** |  |  |
| **8.** | **Mandatory Experiment** | **8** | Snakes and Ladders |  | 21-09-2021 | **1** |  |  |
| **9.** | **Mandatory Experiment** | **9** | Tokenization using NLTK |  | 28-09-2021 | **1** |  |  |
| **10.** | **Mandatory Experiment** | **10** | Knapsack Problem |  | 05-10-2021 | **1** |  |  |
| **11.** | **Mandatory Experiment** |  | **11** | Design an XOR truth table |  | 12-10-2021 | **1** |  |  |
| **12.** | **Performance** | **PR (10)** |  |  |  |  | **10** |  |  |
| **13.** | **Viva** | **Viva (5)** |  |  |  |  | **5** |  |  |

**EXPERIMENT - 1**

# PROGRAM 1

**AIM:** WAP to implement Linear Search

## CODE

def linearSearch(arr, x):

for i in range(len(arr)):

if arr[i] == x:

return i return -1

arr=[]

n = int(input("Enter number of elements : ")) for i in range(0, n):

ele = int(input()) arr.append(ele)

x = int(input("Enter the number to be searched : ")) res = linearSearch(arr,x)

if(res == -1):

print("Number not found") else:

print("Number found at index: ", res)

**OUTPUT**

A screenshot of a computer

Description automatically generated with medium confidence

# PROGRAM 2

**AIM:** WAP to implement Binary Search

## CODE

# **Iterative Method**

def binarySearchIterative(arr, x): l = 0

r = len(arr) - 1 mid = 0

while l <= r:

mid = (r + l) // 2 if arr[mid] < x: l = mid + 1

elif arr[mid] > x: r = mid - 1

else:

return mid

return -1

arr = []

n = int(input("Enter number of elements : ")) for i in range(0, n):

ele = int(input()) arr.append(ele)

x = int(input("Enter the number to be searched : ")) arr.sort()

res = binarySearchIterative(arr, x) if (res == -1):

print("Number not found")

else:

print("Number found at index: ", res)

## OUTPUT

Text

Description automatically generated

**# Recursive**

def binarySearch(arr, l, r, x): if r >= l:

mid = l + (r - l) // 2 if arr[mid] == x:

return mid elif arr[mid] > x:

return binarySearch(arr, l, mid - 1, x) else:

return binarySearch(arr, mid + 1, r, x)

else:

return -1

arr = []

n = int(input("Enter number of elements : ")) for i in range(0, n):

ele = int(input()) arr.append(ele)

x = int(input("Enter the number to be searched : ")) res = binarySearch(arr, 0, len(arr) - 1, x)

if (res == -1):

print("Number not found") else:

print("Number found at index: ", res)

**OUTPUT**

Text

Description automatically generated

# PROGRAM 3

**AIM:** WAP to implement Prims Algorithm

## CODE

INF = 9999999

V = 5

Adj = []

for i in range(0, V):

arr = list(map(int, input().split())) Adj.append(arr)

selected = [0, 0, 0, 0, 0]

no\_edge = 0 selected[0] = True

print("Edge : Weight\n") while (no\_edge < V - 1):

minimum = INF x = 0

y = 0

for i in range(V): if selected[i]:

for j in range(V):

if ((not selected[j]) and Adj[i][j]): if minimum > Adj[i][j]:

minimum = Adj[i][j] x = i

y = j

print(str(x) + "-" + str(y) + ":" + str(Adj[x][y])) selected[y] = True

no\_edge += 1

**OUTPUT**

Text

Description automatically generated

# PROGRAM 4

**AIM:** WAP to implement Kruskal’s Algorithm

## CODE

class Graph:

def init (self, vertices): self.V = vertices self.graph = []

def addEdge(self, u, v, w): self.graph.append([u, v, w])

def find(self, parent, i): if parent[i] == i:

return i

return self.find(parent, parent[i])

def apply\_union(self, parent, rank, x, y): xroot = self.find(parent, x)

yroot = self.find(parent, y) if rank[xroot] < rank[yroot]:

parent[xroot] = yroot

elif rank[xroot] > rank[yroot]: parent[yroot] = xroot

else:

parent[yroot] = xroot rank[xroot] += 1

def kruskal\_algo(self): result = []

i, e = 0, 0

self.graph = sorted(self.graph, key=lambda item: item[2]) parent = []

rank = []

for node in range(self.V): parent.append(node) rank.append(0)

while e < self.V - 1:

u, v, w = self.graph[i] i = i + 1

x = self.find(parent, u) y = self.find(parent, v) if x != y:

e = e + 1 result.append([u, v, w])

self.apply\_union(parent, rank, x, y) for u, v, weight in result:

print("%d - %d: %d" % (u, v, weight))

g = Graph(6)

for i in range(0, 15):

u,v,w = map(int, input().split())

g.addEdge(u, v, w) g.kruskal\_algo()

**OUTPUT**

A picture containing text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT - 2

**AIM:** Write a program to implement BFS for water jug problem using Python

**TOOL USED:** Jupyter Notebook

**THEORY**: Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

Water Jug Problem:

You are given a m liter jug and a n liter jug. Both the jugs are initially empty. The jugs don’t have markings to allow measuring smaller quantities. You have to use the jugs to measure d liters of water where d is less than n.

Production Rules used:

* First we initialize the state and pop off used state, if the state is already visited then continue.
* If the solution state is reached then ans=1, if no solution exists ans=0.
* If we have not reached the final state, we will start developing intermediate states to reach solution state.
* After filling jug 1 and jug 2, and we will check if specified state is possible or not, and accordingly we will empty or fill.
* At the end we will run the function with desired inputs.

## CODE

from collections import deque

def BFS(container\_1, container\_2, target): m = {}

isSolvable = False path = []

q = deque()

q.append((0, 0))

while (len(q) > 0): u = q.popleft()

if ((u[0], u[1]) in m): continue

if ((u[0] > container\_1 or u[1] > container\_2 or u[0] < 0 or u[1] < 0)):

continue path.append([u[0], u[1]])

m[(u[0], u[1])] = 1

if (u[0] == target or u[1] == target): isSolvable = True

if (u[0] == target): if (u[1] != 0):

path.append([u[0], 0])

else:

if (u[0] != 0):

path.append([0, u[1]]) sz = len(path)

for i in range(sz):

print("-->""(", path[i][0], ",",

path[i][1], ")", "-->")

break

q.append([u[0], container\_2]) q.append([container\_1, u[1]])

for ap in range(max(container\_1, container\_2) + 1): c = u[0] + ap

d = u[1] - ap

if (c == container\_1 or (d == 0 and d >= 0)): q.append([c, d])

c = u[0] - ap d = u[1] + ap

if ((c == 0 and c >= 0) or d == container\_2): q.append([c, d])

q.append([container\_1, 0]) q.append([0, container\_2])

if (not isSolvable): print("No solution")

if name == ' main ': J1, J2, target = 4, 3, 2 print("Path is :- \n");

BFS(J1, J2, target)

**OUTPUT**

Text

Description automatically generated with medium confidence

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT - 3

**AIM:** Implement Graph coloring problem using python.

**TOOL USED:** Jupyter Notebook

## THEORY:

Graph Coloring Problem:

Graph coloring problem is to assign colors to certain elements of a graph subject to certain constraints. Vertex coloring is the most common graph coloring problem. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using same color.

## CODE

def printColor(g, V): used = [0] \* V ans = [-1] \* V ans[0] = 0;

for u in range(1, V): for i in g[u]:

if (ans[i] != -1): used[ans[i]] = 1

color = 0

while color < V:

if (used[color] == 0): break

color += 1 ans[u] = color for i in g[u]:

if (ans[i] != -1): used[ans[i]] = 0

maxColor=0

for u in range(V):

print("Vertex", u, " is colored with", ans[u]) if maxColor<ans[u]:

maxColor=ans[u]

print('\nNo of colors used are ',maxColor+1)

def addEdge(g, x, y): g[x].append(y)

g[y].append(x) return g

g = [[] for i in range(6)] g = addEdge(g, 0, 2)

g = addEdge(g, 0, 4)

g = addEdge(g, 0, 1)

g = addEdge(g, 1, 2)

g = addEdge(g, 1, 3)

g = addEdge(g, 1, 5)

g = addEdge(g, 2, 3)

g = addEdge(g, 2, 4)

g = addEdge(g, 3, 5)

g = addEdge(g, 4, 5) printColor(g, 6)

**OUTPUT**

Text

Description automatically generated

# EXPERIMENT - 3(1)

**AIM:** Write a program to implement DFS using Python

**TOOL USED:** Jupyter Notebook

## THEORY:

Depth-first Search (DFS):

Depth-first Search is a tree or graph traversal technique wherein upon reaching a node, the immediately next node traversed is the adjacent node to current node. Thus, the algorithm starts at the root node and explores as far as possible along each branch before backtracking.

## CODE

def dfs(visited, g, src): if visited[src]==0:

print(src, end =" ") visited[src]=1

for nbr in g[src]: dfs(visited, g, nbr)

def addEdge(g, x, y): g[x].append(y)

g[y].append(x) return g

g = [[] for i in range(6)] g = addEdge(g, 0, 2)

g = addEdge(g, 0, 4)

g = addEdge(g, 0, 1)

g = addEdge(g, 1, 2)

g = addEdge(g, 1, 3)

g = addEdge(g, 1, 5)

g = addEdge(g, 2, 3)

g = addEdge(g, 2, 4)

g = addEdge(g, 3, 5)

g = addEdge(g, 4, 5)

visited = [0] \* 6 print("DFS of the graph : ") dfs(visited,g,0)

**OUTPUT**

Text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT - 4

**AIM:** Write a program to implement Best First Search algorithm in python

**TOOL USED:** Jupyter Notebook

## THEORY:

Best First Search:

Best first search is an informed/heuristic method of traversal that decides which node is to be visited nest by checking which node is the most promising one. Graph traversal problem is implemented using priority queue.

Algorithm-

Best-First-Search(Grah g, Node start)

1. Create an empty PriorityQueue PriorityQueue pq;
2. Insert "start" in pq. pq.insert(start)
3. Until PriorityQueue is empty

u = PriorityQueue.DeleteMin If u is the goal

Exit Else

Foreach neighbor v of u If v "Unvisited"

Mark v "Visited" pq.insert(v)

Mark u "Examined"

End procedure

## CODE

from queue import PriorityQueue

def bestFirstSearch(g,src,dest,n,dist): vis = [0] \* n

vis[src] = 1 q=PriorityQueue() q.put((0, src))

while q.empty() == 0: node = q.get()[1] print(node,end=" ") if node == dest:

break

for nbr in g[node]:

if vis[nbr] == 0:

vis[nbr] = 1 q.put((dist[nbr],nbr))

def addEdge(g, x, y): g[x].append(y)

g[y].append(x) return g

g = [[] for i in range(6)] g = addEdge(g, 0, 2)

g = addEdge(g, 0, 4)

g = addEdge(g, 0, 1)

g = addEdge(g, 1, 2)

g = addEdge(g, 1, 3)

g = addEdge(g, 1, 5)

g = addEdge(g, 2, 3)

g = addEdge(g, 2, 4)

g = addEdge(g, 3, 5)

g = addEdge(g, 4, 5) dist=[0,5,4,3,6,0]

bestFirstSearch(g,0,5,6,dist)

**OUTPUT**

A screenshot of a computer screen

Description automatically generated with medium confidence

# EXPERIMENT - 4(1)

**AIM:** Write a program to implement A\* algorithm in python

**TOOL USED:** Jupyter Notebook

## THEORY:

A\* Search:

A-star (also referred to as A\*) is one of the most successful search algorithms to find the shortest path between nodes or graphs. It is an informed search algorithm, as it uses information about path cost and also uses heuristics to find the solution.

## Algorithm-

1: Firstly, Place the starting node into OPEN and find its f (n) value.

2: Then remove the node from OPEN, having the smallest f (n) value. If it is a goal node, then stop and return to success.

3: Else remove the node from OPEN, and find all its successors.

4: Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

5: Goto Step-2.

6: Exit.

## CODE

def aStarAlgo(src, dest): open\_set = set(src) closed\_set = set()

g = {}

parents = {} g[src] = 0 parents[src] = src

while len(open\_set) > 0:

n = None

for v in open\_set:

if n == None or g[v] + heuristic(v) < g[n] + heuristic(n): n = v

if n == dest or graph[n] == None: pass

else:

for (m, weight) in getNeighbors(n):

if m not in open\_set and m not in closed\_set: open\_set.add(m)

parents[m] = n

g[m] = g[n] + weight else:

if g[m] > g[n] + weight: g[m] = g[n] + weight parents[m] = n

if m in closed\_set: closed\_set.remove(m) open\_set.add(m)

if n == None:

print('Path does not exist!') return None

if n == dest: path = []

while parents[n] != n: path.append(n)

n = parents[n] path.append(src) path.reverse()

print('Path found: {}'.format(path)) return path

open\_set.remove(n) closed\_set.add(n)

print('Path does not exist!') return None

def getNeighbors(v): if v in graph:

return graph[v] else:

return None def heuristic(n):

H\_dist = { 'A': 11,

'B': 6,

'C': 99,

'D': 1,

'E': 7,

'G': 0,

}

return H\_dist[n] graph = {

'A': [('B', 2), ('E', 3)],

'B': [('C', 1),('G', 9)],

'C': None, 'E': [('D', 6)],

'D': [('G', 1)],

}

aStarAlgo('A', 'G')

**OUTPUT**

A picture containing text, television, monitor, screen

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT - 5

**AIM:** Write a Program to implement Graph Coloring Problem Constraint Satisfaction Algorithm in python

**TOOL USED:** Jupyter Notebook

## THEORY:

Constraint Satisfaction Algorithm:

In artificial intelligence and operations research, constraint satisfaction is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy. A solution is therefore a set of values for the variables that satisfies all constraints—that is, a point in the feasible region.

## CODE

# Constraints :

# Node 2 and 5 cannot have same color.

# Node 3 cannot be colored with color no. 0

def printColor(g, V): used = [0] \* V ans = [-1] \* V ans[0] = 0;

for u in range(1, V): for i in g[u]:

if (ans[i] != -1): used[ans[i]] = 1

color = 0

while color < V:

if (used[color] == 0 and (u!=5 or ans[2]!=color) and (u!=3 or color!=0)): break

color += 1 ans[u] = color

for i in g[u]:

if (ans[i] != -1): used[ans[i]] = 0

maxColor=0

for u in range(V):

print("Vertex", u, " is colored with", ans[u]) if maxColor<ans[u]:

maxColor=ans[u]

print('\nNo of colors used are ',maxColor+1)

def addEdge(g, x, y): g[x].append(y)

g[y].append(x) return g

g = [[] for i in range(6)] g = addEdge(g, 0, 2)

g = addEdge(g, 0, 4)

g = addEdge(g, 0, 1)

g = addEdge(g, 1, 2)

g = addEdge(g, 1, 3)

g = addEdge(g, 1, 5)

g = addEdge(g, 2, 3)

g = addEdge(g, 2, 4)

g = addEdge(g, 3, 5)

g = addEdge(g, 4, 5)

print("Constraints :\n1)Node 2 and 5 cannot have same color.\n2)Node 3 cannot be colored with color no. 0\n")

printColor(g, 6)

**OUTPUT**

A screenshot of a computer

Description automatically generated with medium confidence

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT - 6

**AIM:** Write a program to implement Tic Tac Toe game problem. Use Min Max Algorithm.

**TOOL USED:** Jupyter Notebook

## THEORY:

Min-max Algorithm:

In artificial intelligence and operations research, Min-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. It performs a depth-first search for the exploration of the complete game tree.

## CODE

playerA, playerB = 'A', 'B'

def isMovesLeft(board) : for i in range(3) :

for j in range(3) :

if (board[i][j] == ' ') :

return True

return False

def evaluate(board) :

for row in range(3) :

if (board[row][0] == board[row][1] and board[row][1] == board[row][2]) :

if (board[row][0] == playerA) : return 1

elif (board[row][0] == playerB) : return -1

for col in range(3) :

if (board[0][col] == board[1][col] and board[1][col] == board[2][col])

:

if (board[0][col] == playerA) : return 1

elif (board[0][col] == playerB) : return -1

if (board[0][0] == board[1][1] and board[1][1] == board[2][2]) : if (board[0][0] == playerA) :

return 1

elif (board[0][0] == playerB) : return -1

if (board[0][2] == board[1][1] and board[1][1] == board[2][0]) : if (board[0][2] == playerA) :

return 1

elif (board[0][2] == playerB) : return -1

return 0

def minimaxAlgo(board, depth, isMax) : score = evaluate(board)

if (score == 1) :

return score if (score == -1) :

return score

if (isMovesLeft(board) == False) : return 0

if (isMax) :

best = -1000

for i in range(3) :

for j in range(3) :

if (board[i][j]==' ') :

board[i][j] = playerA

best = max( best, minimaxAlgo(board,depth +

1,not isMax) )

board[i][j] = ' '

else :

not isMax))

return best

best = 1000

for i in range(3) :

for j in range(3) :

if (board[i][j] == ' ') :

board[i][j] = playerB

best = min(best, minimaxAlgo(board, depth + 1,

board[i][j] = ' '

return best

def findBestMove(board) : bestVal = -1000

bestMove = (-1, -1)

for i in range(3) :

for j in range(3) :

if (board[i][j] == ' ') :

board[i][j] = playerA

moveVal = minimaxAlgo(board, 0, False)

# Backtracking

board[i][j] = ' '

if (moveVal > bestVal) : bestMove = (i, j) bestVal = moveVal

print("The value of the best Move is :", bestVal) return bestMove

board = [

[ 'A', 'B', 'A' ],

[ 'A', ' ', 'A' ],

[ ' ', ' ', 'B' ]

]

bestMove = findBestMove(board)

print("Next best move is :", bestMove[0], bestMove[1])

# Here next turn is of player A

# 0 -> Game will be drawn # 1 -> Player A will win

# -1 -> Player B will win

# -1000 -> No more possible moves exists (To represent this, next best move will be -1,-1)

**OUTPUT**

Text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT – 7 (OPEN-ENDED EXPERIMENT)

**AIM:** Write a program to implement N-Queens problem in Python for 4 Queens (on a 4 × 4 board)

**TOOL USED:** Jupyter Notebook

## THEORY:

N-Queens Problem:

N-Queens problem is a computer problem of placing *n* non-attacking queens on an *n* × *n* chessboard. In chess, a queen attacks any piece which is either vertically or horizontally in line with the queen, or on one of its diagonals.

This problem has a solution for all natural numbers *n* except for *n = 2* and *n = 3*.

## CODE:

global N N = 4

def printSolution(board): for i in range(N):

for j in range(N): print(board[i][j], end=" ")

print()

def isSafe(board, row, col): for i in range(col):

if board[row][i] == 1:

return False

for i, j in zip(range(row, -1, -1),

range(col, -1, -1)): if board[i][j] == 1:

return False

for i, j in zip(range(row, N, 1),

range(col, -1, -1)): if board[i][j] == 1:

return False return True

def solveNQUtil(board, col): if col >= N:

return True

for i in range(N):

if isSafe(board, i, col): board[i][col] = 1

# recur to place rest of the queens

if solveNQUtil(board, col + 1) == True: return True

board[i][col] = 0

return False

def solveNQ():

board = [[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]]

if solveNQUtil(board, 0) == False: print("Solution does not exist") return False

printSolution(board) return True

solveNQ()

**OUTPUT**

Shape

Description automatically generated with medium confidence

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT – 8

**AIM:** Write a program to implement the Single-player game of Snakes and Ladders using Python

**TOOL USED:** Jupyter Notebook

## CODE:

import random

print('Manish Bishnoi \nA2305218529 \n7CSE-8Y') class snakesandladder(object):

def init (self, name, position): self.name = name

self.position = position def dice(self):

chances = 0

print("----------------Let the Games Begin \n")

while self.position <= 100:

roll = random.choice([1, 2, 3, 4, 5, 6]) print('Dice rolls out: ', roll) self.position = roll + self.position

if self.position > 100:

self.position = self.position - roll if self.position == 100:

print('completed the game') break

print('Current position of ' + self.name + ' : ', self.position, '\n') chances += 1

print('Number of turns taken : ', chances)

obj = snakesandladder('Manish', 0) obj.dice()

**OUTPUT**

Text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

# EXPERIMENT – 9

**AIM:** Write a program to implement word tokenization and sentence tokenization of text on Python using the NLTK package

**TOOL USED:** Jupyter Notebook

## THEORY:

Tokenization:

Tokenization refers to the process of breaking unstructured text in natural language into chunks of information that can be considered as discrete elements (or tokens). The token occurrences in a document can be used directly as a vector representing that document.

Tokenization is the first step in any NLP pipeline. It has an important effect on the rest of your pipeline. By turning an unstructured string (text document) into a numerical data structure, it becomes usable for machine learning applications. They might be used in a machine learning pipeline as features that trigger complex decisions or behavior.

Tokenization can be done on the basis of words, sentences, punctuations, or any other meaningful delimiter.

NLTK:

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

## IMPLEMENTATION:

Set-up & Installation:

To install NLTK run the following command on CMD Prompt. pip install nltk

Then download the requisite package from NLTK by running the following code on a cell in Jupyter Notebook.

import nltk nltk.download(‘punkt’)

## Word Tokenization:

import nltk

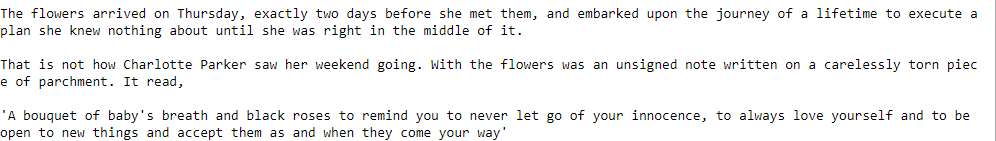
from nltk.tokenize import word\_tokenize

text = "The flowers arrived on Thursday, exactly two days before she met them, and embarked upon the journey of a lifetime to execute a plan she knew nothing about until she was right in the middle of it. \n\nThat is not how Charlotte Parker saw her weekend going. With the flowers was an unsigned note written on a carelessly torn piece of parchment. It read, \n\n'A bouquet of baby's breath and black roses to remind you to never let go of your innocence, to always love yourself and to be open to new things and accept them as and when they come your way'"

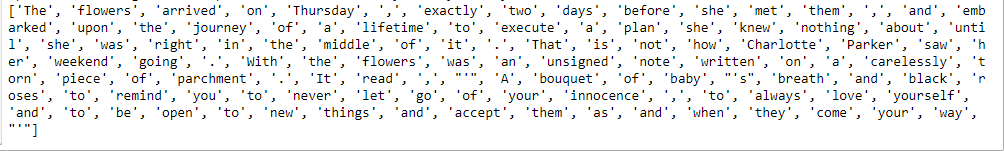
print(text) print(word\_tokenize(text))

## OUTPUT:

Original Text:



Tokenized Text:



1. Sentence Tokenization:

import nltk

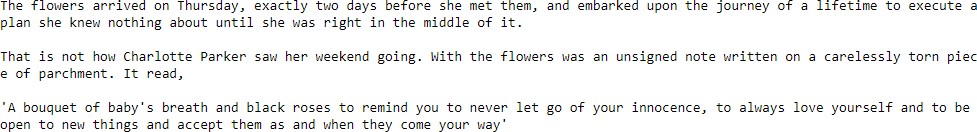
from nltk.tokenize import sent\_tokenize

text = "The flowers arrived on Thursday, exactly two days before she met them, and embarked upon the journey of a lifetime to execute a plan she knew nothing about until she was right in the middle of it. \n\nThat is not how Charlotte Parker saw her weekend going. With the flowers was an unsigned note written on a carelessly torn piece of parchment. It read, \n\n'A bouquet of baby's breath and black roses to remind you to never let go of your innocence, to always love yourself and to be open to new things and accept them as and when they come your way'"

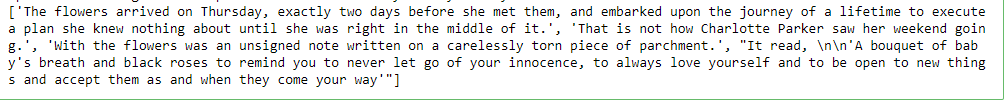
print(text) print(sent\_tokenize(text))

## OUTPUT:

Original Text:



Tokenized Text:



|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

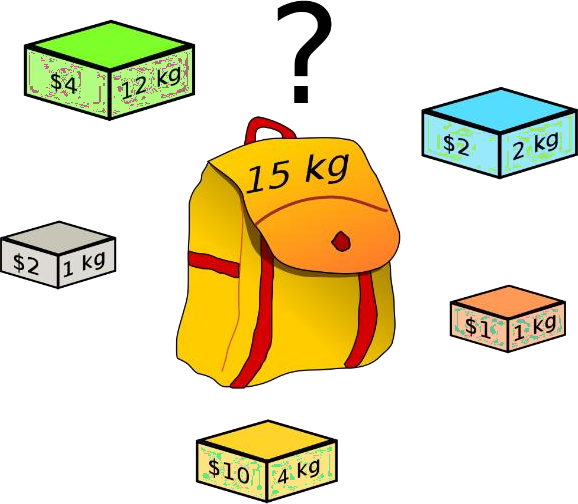
# EXPERIMENT – 10

**AIM:** Write a program to implement the Knapsack problem in Python

**TOOL USED:** Jupyter Notebook

## THEORY:

Knapsack Problem:

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less

than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.

## CODE:

def knapSack(W, wt, val, n): # Base Case

if (n == 0) or (W == 0):

return 0

if (wt[n] > W):

return knapSack(W, wt, val, n - 1)

else:

return max(

val[n] + knapSack(W - wt[n], wt, val, n - 1), knapSack(W, wt, val, n - 1)

)

n = int(input("Enter the number of items available: "))

val = []

wt = []

print("\nEnter value and weight of each item separately:-") for i in range(n):

print("Item " + str(i + 1)) val.append(int(input("Value:")))

wt.append(int(input("Weight:")))

W = int(input("Enter capacity of Knapsack: "))

print("\n\nMaximum profit that can be obtained in given scenario is-") print(knapSack(W, wt, val, n - 1))

print(' \n')

print('Manish Bishnoi \nA2305218529 \n7CSE-8Y')

**OUTPUT:**

Text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

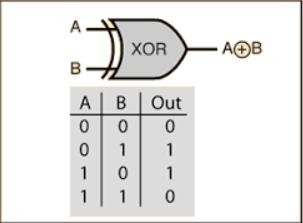
# EXPERIMENT – 11

**AIM:** Write a program to design an XOR truth table using python.

**TOOL USED:** Jupyter Notebook

## THEORY:

XOR” an abbreviation for “Exclusively-OR.” The simplest XOR gate is a two- input digital circuit that outputs a logical “1” if the two input values differ, i.e., its output is a logical “1” if either of its inputs are 1, but not at the same time (exclusively). The symbol and truth table for an XOR is shown in Figure 1



## CODE:

def XOR(a, b): if a != b:

return 1 else:

return 0

if name == ' main ':

print(" | XOR Truth Table | Result |") print(" | | |")

print(" | A = False, B = False | A XOR B =", XOR(False, False), " | ") print(" | A = False, B = True | A XOR B =", XOR(False, True), " | ") print(" | A = True, B = False | A XOR B =", XOR(True, False), " | ") print(" | A = True, B = True | A XOR B =", XOR(True, True), " | ")

print(' ')

print('Manish Bishnoi \nA2305218529 \n7CSE-8Y')

## OUTPUT:

A picture containing text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering**  **ASET, Amity University, Noida (U.P.)** | | | |
| Programme | B. Tech (CSE) | Course Name | Artificial Intelligence |
| Course Code | CSE401 | Semester | 7 |
| Student Name | Madhur Gusain | Enrollment No. | A2305218572 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |